AI32 — Guide to Weka

Andrew Roberts
`http://www.comp.leeds.ac.uk/andyr`

1st March 2005

1 Introduction

Weka is an excellent system for learning about machine learning techniques. Of course, it is a generic toolkit, which means it provides many more features than you will require for your AI32 coursework. Therefore, this guide was written to ensure you know all you require to complete your tasks.

2 ARFF format

For Weka to analyse datasets, it needs a format that users can input so that it understands the structure of the data. The ARFF format is what Weka uses and is very simple. There are only a few tags to be aware of, which all begin with the `@` symbol. Lines beginning with `%` are for comments.

2.1 Header

The header section of an ARFF file is very simple and merely defines the name of the dataset along with the set of attributes and their associated types.

@relation

The `@relation <name>` tag is declared at the beginning of the file, where `<name>` is the name of the relation you wish to use.

@attribute

Attributes are defined in the following format: `@attribute <attribute-name> <type>`. Currently, an attribute can be one of four types:

numeric can be a `real` or `integer` value.

nominal-specification where the value must a from a pre-defined set of possible values.

string textual values.

date [<date-format>] for storing dates. The the optional date format argument instructs Weka on how to parse and print the dates. However, this type will not be very useful for NLP tasks.

Example header:

```
@relation employees



@attribute empName string
@attribute empSalary numeric
@attribute empGender {male, female}
@atttribute empDob date "yyyy-MM-dd"
```

Note also that attributes are case-sensitive, so the label `Name` is different to `name`. It's also worth being aware that Weka can act weirdly when there is a value within the data that is the same as an attribute name. Therefore, it is recommended not to use names that are normal English words to avoid this problem (as you can see in the example, I've added a small prefix to each name.)

2.2 Data

The second half to an ARFF file is the data itself. The `@data` tag signifies to Weka that the instance data is about to commence. Each line after the tag is a set of values (separated by commas) that represent a single instance. It should be obvious that Weka will expect the order of the values to be in the same order in which the attributes were declared.

```
@data
'Andrew Roberts', 50000, male, 1980-11-09
'Phil Space', 20000, male, 1976-04-12
```

Note that string values and nominal values are case sensitive.

3 Getting Weka up and running

1. To load Weka:

   Linux `java -jar /usr/local/weka-3-4/weka.jar`

   Windows I assume there's an icon somewhere!

2. Select *Explorer* on the window that pops up.

3.1 Loading files

By default, the *Preprocess* tab of the Explorer will appear. To load your ARFF file, click *Open File* and use the dialog box to locate your file. If all goes well you should see the window fill up with information about all the attributes in the data file, plus various statistics.

3.2 Selecting features

Depending on the task at hand, it may be the case that you only wish to focus on a subset of the available attributes. On the left hand side of the preprocess window you will see all the attributes. To discard unwanted attributes, you must tick them, and then click the *Remove* button towards the bottom left of the window. Alternatively, it may be quicker to tick the attributes you want to keep, click *Invert*, and then click *Remove*.

3.3 How to classify

Once you are happy with your data and attributes, you can begin experimenting with classifiers. You must click the *Classifier* tab towards the top of the window. At the top left of that window is a button labelled *Choose*, accompanied with a text-box that contains 'ZeroR'. ZeroR is the currently selected classifier, and clicking the button will present a screen which will enable you to select one of the many others available.

ZeroR is not a very useful classifier. Imagine you have a set of instances that can be classified into the classes *Yes* or *No*. ZeroR scans the training data and finds the most frequent class. All future data will be classified according to that most frequent class. It's lack of sophistication, however, makes it a good baseline to compare against. By comparing the accuracy of your (much more sophisticated) classifier versus the ZeroR baseline, you can measure the relative improvement. Of course, if you perform worse than ZeroR, you probably ought to be concerned!

It will be left as an exercise to the reader to investigate which other classifiers Weka has available, and what they all do.

4 Cross-validation

Classifiers rely on being trained before they can reliably be used on new data. Of course, it stands to reason that the more instances the classifier is exposed to during the training phase, the more reliable it will be as it has more experience. However, once trained, we would like to test the classifier too, so that we are confident that it works successfully. For this, yet more unseen instances are required.

A problem which often occurs is the lack of readily available training/test data. These instances must be pre-classified which is typically time-consuming (hence the reason we are trying to automate it with a software classifier!) A nice method to circumvent this issue is know as cross-validation. It works as follows:

1. Separate data in to fixed number of partitions (or folds)
2. Select the first fold for testing, whilst the remaining folds are used for training.
3. Perform classification and obtain performance metrics.
4. Select the next partition as testing and use the rest as training data.
5. Repeat classification until each partition has been used as the test set.
6. Calculate an average performance from the individual experiments.

The experience of many machine learning experiments suggest that using 10 partitions (tenfold cross-validation) often yields the same error rate as if the entire data set had been used for training.

5 Understanding the output

Weka spews all sorts of information after completing the classification task. Some parts are fairly self-explanatory, such as "Correctly Classified Instances", whereas confusion matrices are not necessarily intuitive at first glace and requires some practice to interpret.

5.1 Accuracy

Nothing really difficult here, especially as it's given in the Weka output. It gives a measure for the overall accuracy of the classifier:

$$\text{accuracy} = \frac{\text{number of correctly classified instances}}{\text{number of instances}}$$

5.2 Precision and recall

With respect to classifiers:

$$\text{precision(X)} = \frac{\text{number of correctly classified instances of class X}}{\text{number of instances classified as belonging to class X}}$$

$$\text{recall(X)} = \frac{\text{number of correctly classified instances of class X}}{\text{number of instances in class X}}$$

5.3 Confusion matrix

Confusion matrices are very useful for evaluating classifiers, as they provide an efficient snapshot of its performance — displaying the distribution of correct and incorrect instances. Typical Weka output contains the following:

```
=== Confusion Matrix ===

 a b   <-- classified as
 7 2 | a = yes
 3 2 | b = no
```

Weka was trying to classify instances into two possible classes: *yes* or *no*. For the sake of simplicity, Weka substitutes 'yes' for a, and 'no' for b. The columns represent the instances that were classified as that class. So, the first column shows that in total 10 instances were classified a by Weka, and 4 were classified as b. The rows represent the actual instances that belong to that class. So, what this now tells us is the number of times a given class is correctly/incorrectly classified.

From the matrix, we can observe that 7 of the instances that should have been classed as a, were in fact correctly identified. Similarly, 2 b's were classified correctly. However, we can also see that 2 a's were incorrectly classified as b, where as 3 b's were classed as a. This fine-grained perspective can provide interesting insights. It also allows you to assess the suitability of a particular classifier. Imagine the following classifier that uses a set of attributes to decide whether a patient has cancer or not.

```
=== Confusion Matrix ===

 a  b   <-- classified as
 10 3 | a = cancer
 1  6 | b = no cancer
```

This classifier successfully classifies 16 out of the 20 cases presented. However, an alarming 3 patients will be given the all clear when they did in fact have cancer. The one patient who was told they had cancer despite it being the opposite will also not be happy in the short term, but you can clearly see that the outcome is much more favourable. Now, the classifier was updated, and when fed the same data, it resulted in the following matrix:

```
=== Confusion Matrix ===

 a  b   <-- classified as
 10 0 | a = cancer
 6  4 | b = no cancer
```

This time the classifier does a worse job overall, only correctly classifying 14/20 cases. Yet, it finds no false-negatives. Of course, this is because the system is over cautious and instead labels many false-positives, but it's probably more preferable in this 'better to be safe than sorry' scenario. Hopefully you can now see why confusion matrices are useful for evaluating classifiers beyond a straightforward precision score.

6 Additional resources

- Weka project page — `http://www.cs.waikato.ac.nz/ml/weka/`
- MLnet, the Machine Learning Network Online Information Service — `http://www.mlnet.org/`
- Why using large feature sets can cause problems — `http://en.wikipedia.org/wiki/Curse_of_dimensionality`
- Machine Learning links from the Google Directory — `http://directory.google.com/Top/Computers/Artificial_Intelligence/Machine_Learning`